Lab 25 & 26

**Rules:**

1. You are given “**students.txt**”, a text file that stores a bunch of students. You are also given **TreeMapEx**.**java** that shows how to create and use a Java **TreeMap** container or collection.
2. Do not forget to take your work with you when you leave the lab by either copying your work files to your own USB flash disk, or by e-mailing them to yourself.

In this project you are asked to implement a **Student Information System** application. Your app will enable the user to read in the existing students from a file called “students.txt”, manipulate them and finally store the final student collection back to the file before termination similar to how an information system will behave.

**Student Class**

You will first implement a Student class that consists of four fields: id, firstName, lastName and gpa. The UML class diagram for Student is given below:

A close-up of a student's name

Description automatically generated

Here is the list of methods you must implement for Student:

|  |  |
| --- | --- |
| Student(int id, String firstName, String lastName, double gpa) | Parametrized constructor for Student. Throw an IllegalArgument exception if id <1 or id > 99, firstName or lastName is null or is less than 2 chars long, or gpa is < 0.0 or > 4.0 |
| int getId() | Getter for id |
| Student setId(int id) | Setter for id: Sets the id to the user-supplied value and returns a reference to the object for method chaining. If id < 1 or id > 99, throw an IllegalArgumentException() |
| String getFirstName() | Getter for firstName |
| Student setFirstName(String firstName) | Setter for firstName: Sets the firstName to the user-supplied value and returns a reference to the object for method chaining. If firstName is null or has a length < 2, throw an IllegalArgumentException() |
| String getLastName() | Getter for lastName |
| Student setLastName(String lastName) | Setter for lastName: Sets the lastName to the user-supplied value and returns a reference to the object for method chaining. If lastName is null or has a length < 2, throw an IllegalArgumentException() |
| double getGpa | Getter for gpa |
| Student setGpa(double gpa) | Setter for gpa: Sets gpa to the user-supplied value and returns a reference to the object for method chaining. If gpa < 0 or gpa > 4.0, throw an IllegalArgumentException() |

**students.txt**

This text file contains the current list of students in your database. Here are the contents of the given **students.txt**:

A black background with white text

Description automatically generated

Each line contains one student with different fields separated by a comma (called *comma separated values* – **CSV**). The first column is the id of the student, followed by firstName, lastName and finally the gpa.

**Reading students.txt**

When your application starts running, the first thing it must do is to read this file line-by-line, split each line into separate tokens and create a Student object from each line. You will then insert each student object into a TreeMap as described in the next section.

**TreeMap Collection**

To store the students in memory in sorted order *with respect to their* ***ids***, you will be using the **TreeMap** collection from **java.util**. TreeMap is a standard collection in Java used to store (**key, value**) pairs in ***sorted order*** with respect to the key values. In this application, the **student id** will be used as the key for **TreeMap** and **Student** will be the value associated with the key. You can imagine each key pointing (mapping) to the corresponding Student object as follows:

A diagram of a number

Description automatically generated with medium confidence

Notice that all keys are stored in **sorted order** within the container. How TreeMap stores the (key, value) pairs in sorted order with respect to the key values is the subject of your *Data Structures* course.

To illustrate how you can create a **TreeMap**, insert (key, value) pairs into it, find/delete (key, value) pairs, or iterate over the collection, we have given you **TreeMapEx.java**. Make sure that you look at this example to figure out how to create and use a Java **TreeMap**.

**Student Information System App**

You must implement a main app that starts by creating a TreeMap to store the students and then reads “**students.txt**” to get the current students and populate TreeMap. Your app will then go into a loop, displaying a menu of options that the user can perform. We want your app to have the following 5 menu items as follows:

A black screen with white text

Description automatically generated

Make sure that the user enters a valid choice. Your app must **NOT crash** if the user enters illegal choices. Here is a sample run of our app:

A black screen with white text

Description automatically generated

As you can see, in the first case the user enters **aaa**. In the second case the user enters **0**, and in the third case the user enters **6** all of which are illegal menu items. In each case the app warns the user that an “Illegal menu item is selected” and re-prints the menu. It is only after the user enters a valid option, a number between 1-5, the app continues and performs the necessary action.

**Printing all students**: If the user selects option 1, you must print the existing students in a nicely formatted table. Here is a sample run of our app:

A screen shot of a computer

Description automatically generated

As you can see, we iterated over TreeMap and printed information for each student in a table. Notice that the students are shown in sorted order with respect to their ids. This is made possible by TreeMap.

**Adding a new Student**: If the user selects option 2, you must ask the user to enter the id, firstName, lastName and gpa of the new student, and then add the student to TreeMap. When reading each of the fields from the user, make sure to validate them in a try/catch block. That is, the id entered by the user must be in the range 1-99. You must also make sure that the entered “id” is unique; that is, the entered id must not belong to an existing student. If the id already exists, you must warn the user and continue asking for a valid id. Here is a sample run of our app:

A screen shot of a computer

Description automatically generated

In the first case, the user enters **30**, which already belongs to another student. So the app rejects this id and asks the user to enter a new id. In the second and third cases the user enters **0** and **100** respectively, which are also rejected by the app because the id must be in the range 1-99. In the fourth case, the user enters **fdg**, which is also rejected because the id must be a number. Finally, the user enters 17, which is accepted as a valid id. The app then continues to ask for the firstName.

When reading firstName/lastName, make sure that you trim the possibly extra space chars that may have been entered before or after the actual text and validate that firstName/lastName consists of at least 2 characters. Here is a sample run of our app:

A screen shot of a computer

Description automatically generated

As you can see, the user enters some space chars and then **a** for firstName, which is rejected by the app because firstName must have at least 2 chars. The same kind of validation is applied for lastName.

Finally, ask the user to enter gpa. You must also validate the user-entered gpa to make sure that it is in the range 0.0-4.0. Here is a sample run of our app:

A computer screen shot of a number

Description automatically generated

After the user enters valid values for all fields, you must create a new student and add it to TreeMap. Also print the information about the newly added student as shown above.

When you now print all students by selecting option 1 from the main menu, you will see that the new student appears in sorted order among the existing students as follows:

A screen shot of a computer

Description automatically generated

**Deleting an existing student**: If the user selects option 3, ask the user to enter the id of the student and delete it from TreeMap if it exists. If the user does not exist in TreeMap, do nothing. Here is a sample run of our app:

A screen shot of a computer

Description automatically generated

**Finding an existing student**: If the user selects option 4, ask the user to enter the id of the student and print its details on the screen if the student exists. If the student does not exist, then print a message saying that the user does not exist. Here is a sample run of our app:

A screenshot of a computer

Description automatically generated

**Exiting the app**: If the user selects option 5, you must exit the app. Before you exit though, **you must save the existing students** to “**students.txt**” file. In our sample run so far, we deleted student with id=15 and added a student with id=17. So after our app terminates, here are the contents of “students.txt”. Notice that because we are iterating over the TreeMap, students appear in sorted order wrt their key values:

A black background with white text

Description automatically generated

Lab Work Submission:

* You can continue to work on this lab after our lab class, on your own, at home.
* Submit your lab work via Blackboard on or before: **Wednesday, December 6, 2023, 11:59pm**.
* The only accepted submission method!
* Once you submit your assignment you will not be able to resubmit it!
* Make absolutely sure the Java files you want to submit are the Java files you want graded.
* You will not be able to submit your lab work under any circumstances once **Lab25** disappears at **12:00 a.m.** on **Thursday, December 7, 2023**.
* There will be **NO** exceptions to these rules!
* To submit your lab work, upload all files you did for this lab (.java extension) to the **Lab25** assignment in the **Labs** tab in your Lab section’s presence in Blackboard.
* Then, make sure you click the **Submit** button to submit your lab work.
* This lab is worth **20 points**.